P 53

53.1 PACKSSWB/PACKSSDW—Pack with Signed Saturation

Opcode	Instruction	Description
0F 63 /r	PACKSSWB *mm, mm/m64*	Packs and saturate pack 4 signed words from *mm* and 4 signed words from *mm/m64* into 8 signed bytes in *mm.*
0F 6B /r	PACKSSDW *mm, mm/m64*	Pack and saturate 2 signed doublewords from *mm* and 2 signed doublewords from *mm/m64* into 4 signed words in *mm.*

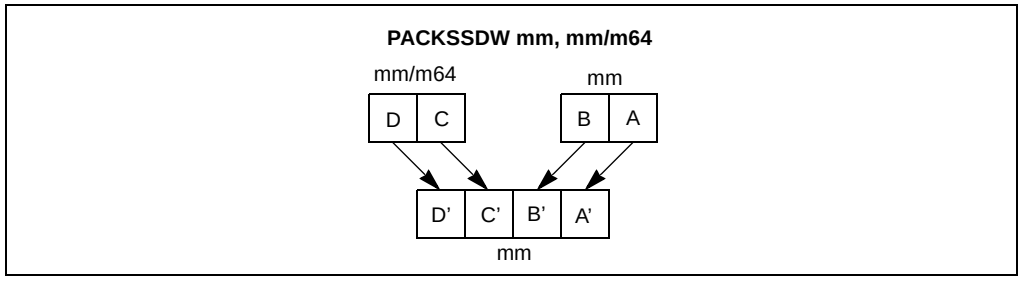
Description

Packs and saturates signed words into bytes (PACKSSWB) or signed doublewords into words (PACKSSDW). The PACKSSWB instruction packs 4 signed words from the destination operand (first operand) and 4 signed words from the source operand (second operand) into 8 signed bytes in the destination operand. If the signed value of a word is beyond the range of a signed byte (that is, greater than 7FH or less than 80H), the saturated byte value of 7FH or 80H, respectively, is stored into the destination.

The PACKSSDW instruction packs 2 signed doublewords from the destination operand (first operand) and 2 signed doublewords from the source operand (second operand) into 4 signed words in the destination operand (see Figure 53-1). If the signed value of a doubleword is beyond the range of a signed word (that is, greater than 7FFFH or less than 8000H), the saturated word value of 7FFFH or 8000H, respectively, is stored into the destination.

The destination operand for either the PACKSSWB or PACKSSDW instruction must be an MMX register; the source operand may be either an MMX register or a quadword memory location.

Figure 53-1. Operation of the PACKSSDW Instruction



Operation

```
IF instruction is PACKSSWB
    THEN
        DEST(7..0) ← SaturateSignedWordToSignedByte DEST(15..0);
        DEST(15..8) ← SaturateSignedWordToSignedByte DEST(31..16);
        DEST(23..16) ← SaturateSignedWordToSignedByte DEST(47..32);
        DEST(31..24) ← SaturateSignedWordToSignedByte DEST(63..48);
        DEST(39..32) ← SaturateSignedWordToSignedByte SRC(15..0);
        DEST(47..40) ← SaturateSignedWordToSignedByte SRC(31..16);
        DEST(55..48) ← SaturateSignedWordToSignedByte SRC(47..32);
        DEST(63..56) ← SaturateSignedWordToSignedByte SRC(63..48);
```

```
    ELSE (* instruction is PACKSSDW *)
        DEST(15..0) ← SaturateSignedDoublewordToSignedWord DEST(31..0);
        DEST(31..16) ← SaturateSignedDoublewordToSignedWord DEST(63..32);
        DEST(47..32) ← SaturateSignedDoublewordToSignedWord SRC(31..0);
        DEST(63..48) ← SaturateSignedDoublewordToSignedWord SRC(63..32);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.2 PACKUSWB—Pack with Unsigned Saturation

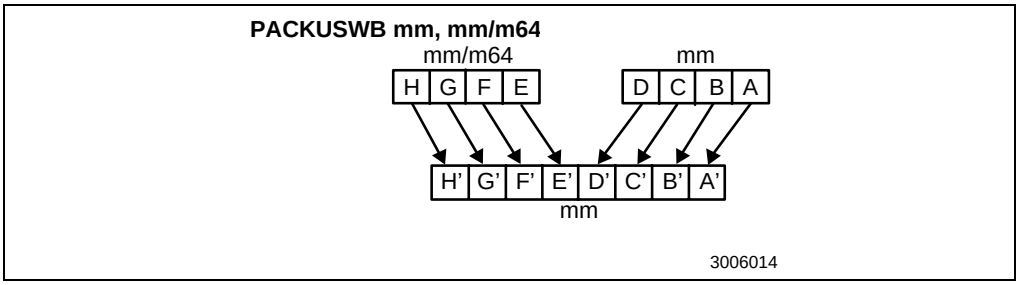
Opcode	Instruction	Description
0F 67 /r	PACKUSWB *mm, mm/m64*	Pack and saturate 4 signed words from *mm* and 4 signed words from *mm/m64* into 8 unsigned bytes in *mm*.

Description

Packs and saturates 4 signed words from the destination operand (first operand) and 4 signed words from the source operand (second operand) into 8 unsigned bytes in the destination operand (see Figure 53-2). If the signed value of a word is beyond the range of an unsigned byte (that is, greater than FFH or less than 00H), the saturated byte value of FFH or 00H, respectively, is stored into the destination.

The destination operand must be an MMX register; the source operand may be either an MMX register or a quadword memory location.

Figure 53-2. Operation of the PACKUSWB Instruction



Operation

```
DEST(7..0) ← SaturateSignedWordToUnsignedByte DEST(15..0);
DEST(15..8) ← SaturateSignedWordToUnsignedByte DEST(31..16);
DEST(23..16) ← SaturateSignedWordToUnsignedByte DEST(47..32);
DEST(31..24) ← SaturateSignedWordToUnsignedByte DEST(63..48);
DEST(39..32) ← SaturateSignedWordToUnsignedByte SRC(15..0);
DEST(47..40) ← SaturateSignedWordToUnsignedByte SRC(31..16);
DEST(55..48) ← SaturateSignedWordToUnsignedByte SRC(47..32);
DEST(63..56) ← SaturateSignedWordToUnsignedByte SRC(63..48);
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.

#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.3 PADDB/PADDW/PADDD—Packed Add

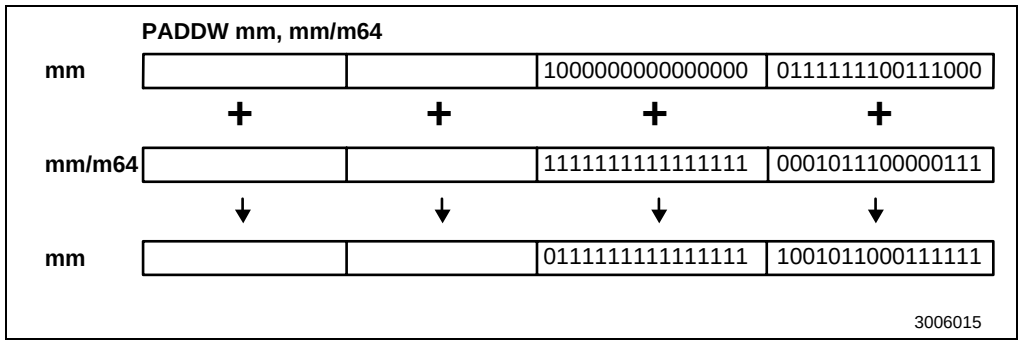
Opcode	Instruction	Description
0F FC /r	PADDB *mm, mm/m64*	Add packed bytes from *mm/m64* to packed bytes in *mm.*
0F FD /r	PADDW *mm, mm/m64*	Add packed words from *mm/m64* to packed words in *mm.*
0F FE /r	PADDD *mm, mm/m64*	Add packed doublewords from *mm/m64* to packed doublewords in *mm.*

Description

Adds the individual data elements (bytes, words, or doublewords) of the source operand (second operand) to the individual data elements of the destination operand (first operand). (See Figure 53-3.) If the result of an individual addition exceeds the range for the specified data type (overflows), the result is wrapped around, meaning that the result is truncated so that only the lower (least significant) bits of the result are returned (that is, the carry is ignored).

The destination operand must be an MMX register; the source operand can be either an MMX register or a quadword memory location.

Figure 53-3. Operation of the PADDW Instruction



The PADDB instruction adds the bytes of the source operand to the bytes of the destination operand and stores the results to the destination operand. When an individual result is too large to be represented in 8 bits, the lower 8 bits of the result are written to the destination operand and therefore the result wraps around.

The PADDW instruction adds the words of the source operand to the words of the destination operand and stores the results to the destination operand. When an individual result is too large to be represented in 16 bits, the lower 16 bits of the result are written to the destination operand and therefore the result wraps around.

The PADDD instruction adds the doublewords of the source operand to the doublewords of the destination operand and stores the results to the destination operand. When an individual result is too large to be represented in 32 bits, the lower 32 bits of the result are written to the destination operand and therefore the result wraps around.

Note that like the integer ADD instruction, the PADDB, PADDW, and PADDD instructions can operate on either unsigned or signed (two's complement notation) packed integers. Unlike the integer instructions, none of the MMX instructions affect the EFLAGS register. With MMX instructions, there are no carry or overflow flags to indicate when overflow has occurred, so the software must control the range of values or else use the "with saturation" MMX instructions.

Operation

```
IF instruction is PADDB
    THEN
        DEST(7..0) ← DEST(7..0) + SRC(7..0);
        DEST(15..8) ← DEST(15..8) + SRC(15..8);
        DEST(23..16) ← DEST(23..16)+ SRC(23..16);
        DEST(31..24) ← DEST(31..24) + SRC(31..24);
        DEST(39..32) ← DEST(39..32) + SRC(39..32);
        DEST(47..40) ← DEST(47..40)+ SRC(47..40);
        DEST(55..48) ← DEST(55..48) + SRC(55..48);
        DEST(63..56) ← DEST(63..56) + SRC(63..56);
ELSEIF instruction is PADDW
    THEN
        DEST(15..0) ← DEST(15..0) + SRC(15..0);
        DEST(31..16) ← DEST(31..16) + SRC(31..16);
        DEST(47..32) ← DEST(47..32) + SRC(47..32);
        DEST(63..48) ← DEST(63..48) + SRC(63..48);
    ELSE (* instruction is PADDD *)
        DEST(31..0) ← DEST(31..0) + SRC(31..0);
        DEST(63..32) ← DEST(63..32) + SRC(63..32);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

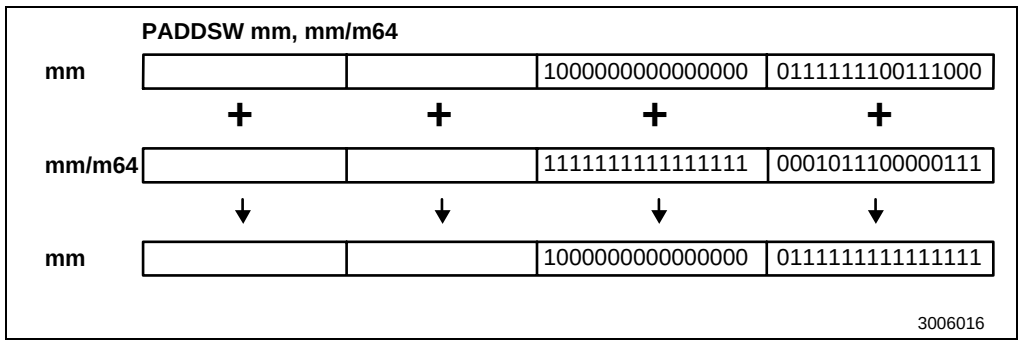
53.4 PADDSB/PADDSW—Packed Add with Saturation

Opcode	Instruction	Description
0F EC /r	PADDSB *mm, mm/m64*	Add signed packed bytes from *mm/m64* to signed packed bytes in *mm* and saturate.
0F ED /r	PADDSW *mm, mm/m64*	Add signed packed words from *mm/m64* to signed packed words in *mm* and saturate.

Description

Adds the individual signed data elements (bytes or words) of the source operand (second operand) to the individual signed data elements of the destination operand (first operand). (See Figure 53-4.) If the result of an individual addition exceeds the range for the specified data type, the result is saturated. The destination operand must be an MMX register; the source operand can be either an MMX register or a quadword memory location.

Figure 53-4. Operation of the PADDSW Instruction



The PADDSB instruction adds the signed bytes of the source operand to the signed bytes of the destination operand and stores the results to the destination operand. When an individual result is beyond the range of a signed byte (that is, greater than 7FH or less than 80H), the saturated byte value of 7FH or 80H, respectively, is written to the destination operand.

The PADDSW instruction adds the signed words of the source operand to the signed words of the destination operand and stores the results to the destination operand. When an individual result is beyond the range of a signed word (that is, greater than 7FFFH or less than 8000H), the saturated word value of 7FFFH or 8000H, respectively, is written to the destination operand.

Operation

```
IF instruction is PADDSB
    THEN
        DEST(7..0) ← SaturateToSignedByte(DEST(7..0) + SRC (7..0)) ;
        DEST(15..8) ← SaturateToSignedByte(DEST(15..8) + SRC(15..8) );
        DEST(23..16) ← SaturateToSignedByte(DEST(23..16)+ SRC(23..16) );
        DEST(31..24) ← SaturateToSignedByte(DEST(31..24) + SRC(31..24) );
        DEST(39..32) ← SaturateToSignedByte(DEST(39..32) + SRC(39..32) );
        DEST(47..40) ← SaturateToSignedByte(DEST(47..40)+ SRC(47..40) );
        DEST(55..48) ← SaturateToSignedByte(DEST(55..48) + SRC(55..48) );
        DEST(63..56) ← SaturateToSignedByte(DEST(63..56) + SRC(63..56) );
    ELSE { (* instruction is PADDSW *)
        DEST(15..0) ← SaturateToSignedWord(DEST(15..0) + SRC(15..0) );
        DEST(31..16) ← SaturateToSignedWord(DEST(31..16) + SRC(31..16) );
        DEST(47..32) ← SaturateToSignedWord(DEST(47..32) + SRC(47..32) );
        DEST(63..48) ← SaturateToSignedWord(DEST(63..48) + SRC(63..48) );
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.

#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

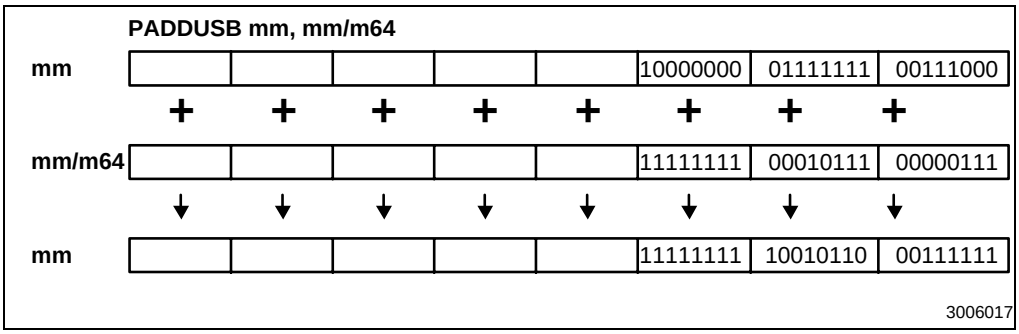
53.5 PADDUSB/PADDUSW—Packed Add Unsigned with Saturation

Opcode	Instruction	Description
0F DC /r	PADDUSB *mm, mm/m64*	Add unsigned packed bytes from *mm/m64* to unsigned packed bytes in *mm* and saturate.
0F DD /r	PADDUSW *mm, mm/m64*	Add unsigned packed words from *mm/m64* to unsigned packed words in *mm* and saturate.

Description

Adds the individual unsigned data elements (bytes or words) of the packed source operand (second operand) to the individual unsigned data elements of the packed destination operand (first operand). (See Figure 53-5.) If the result of an individual addition exceeds the range for the specified unsigned data type, the result is saturated. The destination operand must be an MMX register; the source operand can be either an MMX register or a quadword memory location.

Figure 53-5. Operation of the PADDUSB Instruction



The PADDUSB instruction adds the unsigned bytes of the source operand to the unsigned bytes of the destination operand and stores the results to the destination operand. When an individual result is beyond the range of an unsigned byte (that is, greater than FFH), the saturated unsigned byte value of FFH is written to the destination operand.

The PADDUSW instruction adds the unsigned words of the source operand to the unsigned words of the destination operand and stores the results to the destination operand. When an individual result is beyond the range of an unsigned word (that is, greater than FFFFH), the saturated unsigned word value of FFFFH is written to the destination operand.

Operation

```
IF instruction is PADDUSB
    THEN
        DEST(7..0) ← SaturateToUnsignedByte(DEST(7..0) + SRC (7..0) );
        DEST(15..8) ← SaturateToUnsignedByte(DEST(15..8) + SRC(15..8) );
        DEST(23..16) ← SaturateToUnsignedByte(DEST(23..16)+ SRC(23..16) );
        DEST(31..24) ← SaturateToUnsignedByte(DEST(31..24) + SRC(31..24) );
        DEST(39..32) ← SaturateToUnsignedByte(DEST(39..32) + SRC(39..32) );
        DEST(47..40) ← SaturateToUnsignedByte(DEST(47..40)+ SRC(47..40) );
        DEST(55..48) ← SaturateToUnsignedByte(DEST(55..48) + SRC(55..48) );
        DEST(63..56) ← SaturateToUnsignedByte(DEST(63..56) + SRC(63..56) );
    ELSE { (* instruction is PADDUSW *)
        DEST(15..0) ← SaturateToUnsignedWord(DEST(15..0) + SRC(15..0) );
        DEST(31..16) ← SaturateToUnsignedWord(DEST(31..16) + SRC(31..16) );
        DEST(47..32) ← SaturateToUnsignedWord(DEST(47..32) + SRC(47..32) );
        DEST(63..48) ← SaturateToUnsignedWord(DEST(63..48) + SRC(63..48) );
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.

#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

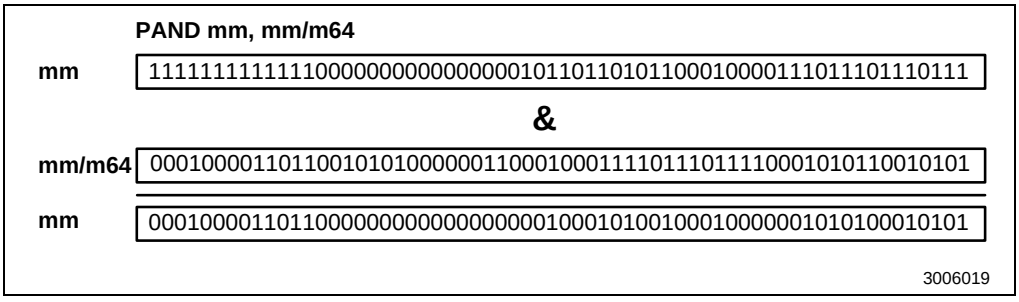
53.6 PAND—Logical AND

Opcode	Instruction	Description
0F DB /r	PAND *mm, mm/m64*	AND quadword from *mm/m64* to quadword in mm.

Description

Performs a bitwise logical AND operation on the quadword source (second) and destination (first) operands and stores the result in the destination operand location (see Figure 53-6). The source operand can be an MMX register or a quadword memory location; the destination operand must be an MMX register. Each bit of the result of the PAND instruction is set to 1 if the corresponding bits of the operands are both 1; otherwise it is made zero

Figure 53-6. Operation of the PAND Instruction



Operation

```
DEST ← DEST AND SRC;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.7 PANDN—Logical AND NOT

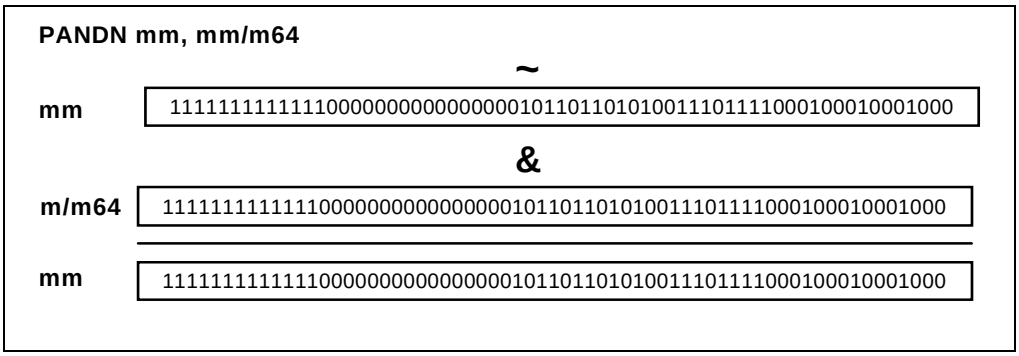
Opcode	Instruction	Description
0F DF /r	PANDN *mm, mm/m64*	AND quadword from *mm/m64* to NOT quadword in *mm.*

Description

Performs a bitwise logical NOT on the quadword destination operand (first operand). Then, the instruction performs a bitwise logical AND operation on the inverted destination operand and the quadword source operand (second operand). (See Figure 53-7.) Each bit of the result of the AND operation is set to one if the corresponding bits of the source and inverted destination bits are one; otherwise it is set to zero. The result is stored in the destination operand location.

The source operand can be an MMX register or a quadword memory location; the destination operand must be an MMX register.

Figure 53-7. Operation of the PANDN Instruction



Operation

```
DEST ← (NOT DEST) AND SRC;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

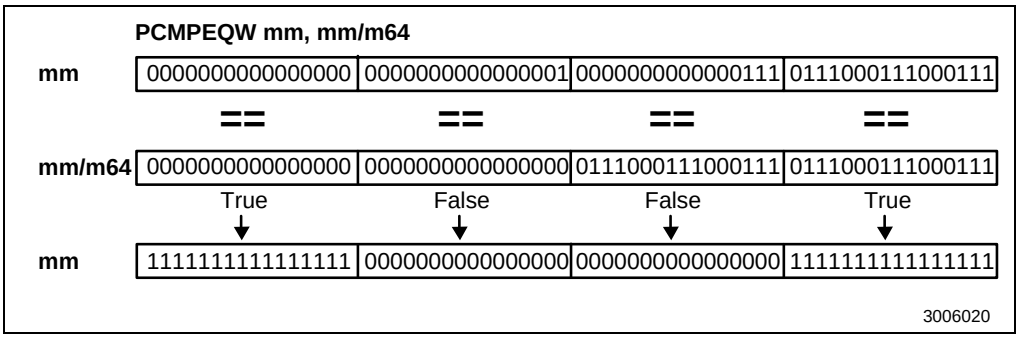
53.8 PCMPEQB/PCMPEQW/PCMPEQD—Packed Compare for Equal

Opcode	Instruction	Description
0F 74 /r	PCMPEQB *mm, mm/m64*	Compare packed bytes in *mm/m64* with packed bytes in *mm* for equality.
0F 75 /r	PCMPEQW *mm, mm/m64*	Compare packed words in *mm/m64* with packed words in *mm* for equality.
0F 76 /r	PCMPEQD *mm, mm/m64*	Compare packed doublewords in *mm/m64* with packed doublewords in *mm* for equality.

Description

Compares the individual data elements (bytes, words, or doublewords) in the destination operand (first operand) to the corresponding data elements in the source operand (second operand). (See Figure 53-8.) If a pair of data elements are equal, the corresponding data element in the destination operand is set to all ones; otherwise, it is set to all zeros. The destination operand must be an MMX register; the source operand may be either an MMX register or a 64-bit memory location.

Figure 53-8. Operation of the PCMPEQW Instruction



The PCMPEQB instruction compares the bytes in the destination operand to the corresponding bytes in the source operand, with the bytes in the destination operand being set according to the results.

The PCMPEQW instruction compares the words in the destination operand to the corresponding words in the source operand, with the words in the destination operand being set according to the results.

The PCMPEQD instruction compares the doublewords in the destination operand to the corresponding doublewords in the source operand, with the doublewords in the destination operand being set according to the results.

Operation

```
IF instruction is PCMPEQB
    THEN
        IF DEST(7..0) = SRC(7..0)
            THEN DEST(7  0) ← FFH;
            ELSE DEST(7..0) ← 0;
        * Continue comparison of second through seventh bytes in DEST and SRC *
        IF DEST(63..56) = SRC(63..56)
            THEN DEST(63..56) ← FFH;
            ELSE DEST(63..56) ← 0;
ELSE IF instruction is PCMPEQW
    THEN
        IF DEST(15..0) = SRC(15..0)
            THEN DEST(15..0) ← FFFFH;
            ELSE DEST(15..0) ← 0;
        * Continue comparison of second and third words in DEST and SRC *
        IF DEST(63..48) = SRC(63..48)
            THEN DEST(63..48) ← FFFFH;
            ELSE DEST(63..48) ← 0;
    ELSE (* instruction is PCMPEQD *)
        IF DEST(31..0) = SRC(31..0)
            THEN DEST(31..0) ← FFFFFFFFH;
            ELSE DEST(31..0) ← 0;
        IF DEST(63..32) = SRC(63..32)
            THEN DEST(63..32) ← FFFFFFFFH;
            ELSE DEST(63..32) ← 0;
FI;
```

Flags Affected

None:

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.

#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.9 PCMPGTB/PCMPGTW/PCMPGTD—Packed Compare for Greater Than

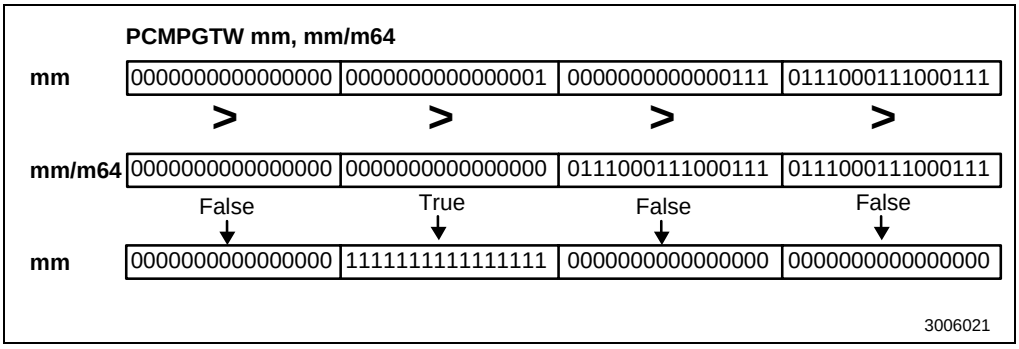
Opcode	Instruction	Description
0F 64 /r	PCMPGTB *mm, mm/ m64*	Compare packed bytes in *mm* with packed bytes in *mm/m64* for greater value.
0F 65 /r	PCMPGTW *mm, mm/ m64*	Compare packed words in *mm* with packed words in *mm/m64* for greater value.
0F 66 /r	PCMPGTD *mm, mm/ m64*	Compare packed doublewords in *mm* with packed doublewords in *mm/m64* for greater value.

Description

Compare the individual signed data elements (bytes, words, or doublewords) in the destination operand (first operand) to the corresponding signed data elements in the source operand (second operand). (See Figure 53-9.) If a data element in the destination operand is greater than its

corresponding data element in the source operand, the data element in the destination operand is set to all ones; otherwise, it is set to all zeros. The destination operand must be an MMX register; the source operand may be either an MMX register or a 64-bit memory location.

Figure 53-9. Operation of the PCMPGTW Instruction



The PCMPGTB instruction compares the signed bytes in the destination operand to the corresponding signed bytes in the source operand, with the bytes in the destination operand being set according to the results.

The PCMPGTW instruction compares the signed words in the destination operand to the corresponding signed words in the source operand, with the words in the destination operand being set according to the results.

The PCMPGTD instruction compares the signed doublewords in the destination operand to the corresponding signed doublewords in the source operand, with the doublewords in the destination operand being set according to the results.

Operation

```
IF instruction is PCMPGTB
     THEN
          IF DEST(7..0) > SRC(7..0)
               THEN DEST(7  0) ← FFH;
               ELSE DEST(7..0) ← 0;
          * Continue comparison of second through seventh bytes in DEST and SRC *
          IF DEST(63..56) > SRC(63..56)
               THEN DEST(63..56) ← FFH;
               ELSE DEST(63..56) ← 0;
ELSE IF instruction is PCMPGTW
     THEN
          IF DEST(15..0) > SRC(15..0)
               THEN DEST(15..0) ← FFFFH;
               ELSE DEST(15..0) ←0;
          * Continue comparison of second and third bytes in DEST and SRC *
          IF DEST(63..48) > SRC(63..48)
               THEN DEST(63..48) ← FFFFH;
               ELSE DEST(63..48) ← 0;
     ELSE { (* instruction is PCMPGTD *)
          IF DEST(31..0) > SRC(31..0)
               THEN DEST(31..0) ← FFFFFFFFH;
               ELSE DEST(31..0) ← 0;
          IF DEST(63..32) > SRC(63..32)
               THEN DEST(63..32) ← FFFFFFFFH;
               ELSE DEST(63..32) ← 0;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.10 PMADDWD—Packed Multiply and Add

Opcode	Instruction	Description
0F F5 /r	PMADDWD *mm, mm/ m64*	Multiply the packed words in *mm* by the packed words in *mm/m64*. Add the 32-bit pairs of results and store in *mm* as doubleword

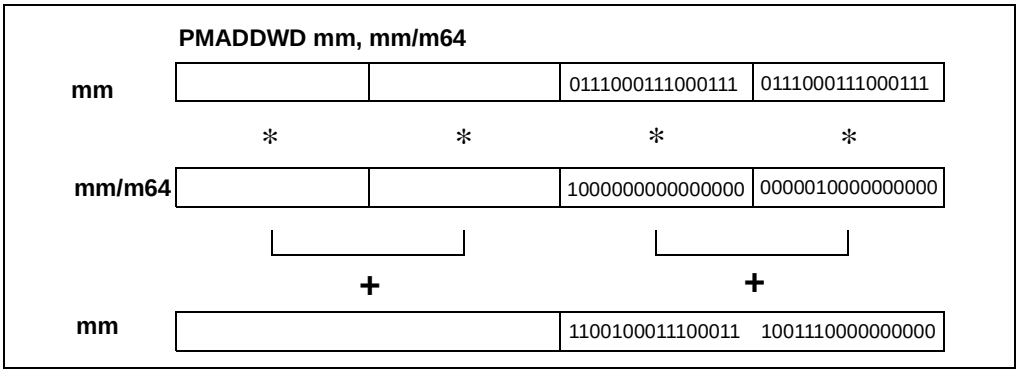
Description

Multiplies the individual signed words of the destination operand by the corresponding signed words of the source operand, producing four signed, doubleword results (see Figure 53-10). The two doubleword results from the multiplication of the high-order words are added together and stored in the upper doubleword of the destination operand; the two doubleword results from the

multiplication of the low-order words are added together and stored in the lower doubleword of the destination operand. The destination operand must be an MMX register; the source operand may be either an MMX register or a 64-bit memory location.

The PMADDWD instruction wraps around to 80000000H only when all four words of both the source and destination operands are 8000H.

Figure 53-10. Operation of the PMADDWD Instruction



Operation

```
DEST(31..0) ← (DEST(15..0) * SRC(15..0)) + (DEST(31..16) * SRC(31..16));
DEST(63..32) ← (DEST(47..32) * SRC(47..32)) + (DEST(63..48) * SRC(63..48));
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

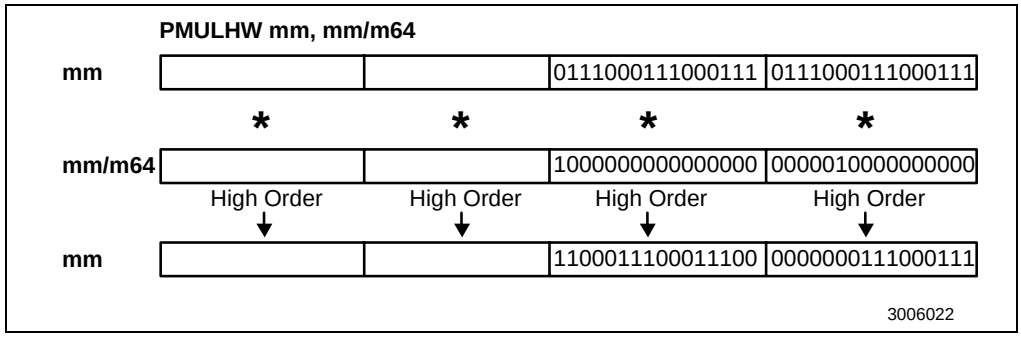
53.11 PMULHW—Packed Multiply High

Opcode	Instruction	Description
0F E5 /r	PMULHW *mm, mm/m64*	Multiply the signed packed words in *mm* by the signed packed words in *mm/m64*, then store the high-order word of each doubleword result in *mm*.

Description

Multiplies the four signed words of the source operand (second operand) by the four signed words of the destination operand (first operand), producing four signed, doubleword, intermediate results (see Figure 53-11). The high-order word of each intermediate result is then written to its corresponding word location in the destination operand. The destination operand must be an MMX register; the source operand may be either an MMX register or a 64-bit memory location.

Figure 53-11. Operation of the PMULHW Instruction



Operation

```
DEST(15..0) ← HighOrderWord(DEST(15..0) * SRC(15..0));
DEST(31..16) ← HighOrderWord(DEST(31..16) * SRC(31..16));
DEST(47..32) ← HighOrderWord(DEST(47..32) * SRC(47..32));
DEST(63..48) ← HighOrderWord(DEST(63..48) * SRC(63..48));
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.12 PMULLW—Packed Multiply Low

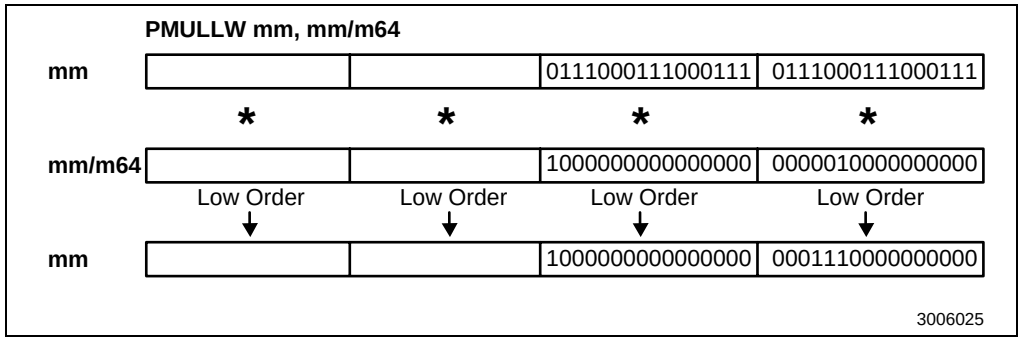
Opcode	Instruction	Description
0F D5 /r	PMULLW *mm, mm/ m64*	Multiply the packed words in *mm* with the packed words in *mm/m64*, then store the low-order word of each doubleword result in *mm*.

Description

Multiplies the four signed or unsigned words of the source operand (second operand) with the four signed or unsigned words of the destination operand (first operand), producing four doubleword, intermediate results (see Figure 53-12). The low-order word of each intermediate result is then

written to its corresponding word location in the destination operand. The destination operand must be an MMX register; the source operand may be either an MMX register or a 64-bit memory location.

Figure 53-12. Operation of the PMULLW Instruction



Operation

```
DEST(15..0) ← LowOrderWord(DEST(15..0) * SRC(15..0));
DEST(31..16) ← LowOrderWord(DEST(31..16) * SRC(31..16));
DEST(47..32) ← LowOrderWord(DEST(47..32) * SRC(47..32));
DEST(63..48) ← LowOrderWord(DEST(63..48) * SRC(63..48));
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.13 POP—Pop a Value from the Stack

Opcode	Instruction	Description
8F /0	POP *m16*	Pop top of stack into *m16*; increment stack pointer
8F /0	POP *m32*	Pop top of stack into *m32*; increment stack pointer
58+ *rw*	POP *r16*	Pop top of stack into *r16*; increment stack pointer
58+ *rd*	POP *r32*	Pop top of stack into *r32*; increment stack pointer
1F	POP DS	Pop top of stack into DS; increment stack pointer
07	POP ES	Pop top of stack into ES; increment stack pointer
17	POP SS	Pop top of stack into SS; increment stack pointer
0F A1	POP FS	Pop top of stack into FS; increment stack pointer
0F A9	POP GS	Pop top of stack into GS; increment stack pointer

Description

Loads the value from the top of the stack to the location specified with the destination operand and then increments the stack pointer. The destination operand can be a general-purpose register, memory location, or segment register.

The address-size attribute of the stack segment determines the stack pointer size (16 bits or 32 bits—the source address size), and the operand-size attribute of the current code segment determines the amount the stack pointer is incremented (2 bytes or 4 bytes). For example, if these address- and operand-size attributes are 32, the 32-bit ESP register (stack pointer) is incremented by 4 and, if they are 16, the 16-bit SP register is incremented by 2. (The B flag in the stack segment's segment descriptor determines the stack's address-size attribute, and the D flag in the current code segment's segment descriptor, along with prefixes, determines the operand-size attribute and also the address-size attribute of the destination operand.)

If the destination operand is one of the segment registers DS, ES, FS, GS, or SS, the value loaded into the register must be a valid segment selector. In protected mode, popping a segment selector into a segment register automatically causes the descriptor information associated with that segment selector to be loaded into the hidden (shadow) part of the segment register and causes the selector and the descriptor information to be validated (see the "Operation" section below).

A null value (0000-0003) may be popped into the DS, ES, FS, or GS register without causing a general protection fault. However, any subsequent attempt to reference a segment whose corresponding segment register is loaded with a null value causes a general protection exception (#GP). In this situation, no memory reference occurs and the saved value of the segment register is null.

The POP instruction cannot pop a value into the CS register. To load the CS register from the stack, use the RET instruction.

If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register. In the case of a 16-bit stack where ESP wraps to 0H as a result of the POP instruction, the resulting location of the memory write is processor family specific.

The POP ESP instruction increments the stack pointer (ESP) before data at the old top of stack is written into the destination.

A POP SS instruction inhibits all interrupts, including the NMI interrupt, until after execution of the next instruction. This action allows sequential execution of POP SS and MOV ESP, EBP instructions without the danger of having an invalid stack during an interrupt[1]. However, use of the LSS instruction is the preferred method of loading the SS and ESP registers.

Operation

```
IF StackAddrSize = 32
    THEN
        IF OperandSize = 32
            THEN
                DEST ← SS:ESP; (* copy a doubleword *)
                ESP ← ESP + 4;
            ELSE (* OperandSize = 16*)
                DEST ← SS:ESP; (* copy a word *)
            ESP ← ESP + 2;
        FI;
    ELSE (* StackAddrSize = 16* )
        IF OperandSize = 16
            THEN
                DEST ← SS:SP; (* copy a word *)
                SP ← SP + 2;
            ELSE (* OperandSize = 32 *)
                DEST ← SS:SP; (* copy a doubleword *)
                SP ← SP + 4;
        FI;
FI;
```

Loading a segment register while in protected mode results in special checks and actions, as described in the following listing. These checks are performed on the segment selector and the segment descriptor it points to.

```
IF SS is loaded;
    THEN
        IF segment selector is null
            THEN #GP(0);
        FI;
        IF segment selector index is outside descriptor table limits
            OR segment selector's RPL ≠ CPL
            OR segment is not a writable data segment
            OR DPL ≠ CPL
                THEN #GP(selector);
        FI;
        IF segment not marked present
```

1. Note that in a sequence of instructions that individually delay interrupts past the following instruction, only the first instruction in the sequence is guaranteed to delay the interrupt, but subsequent interrupt-delaying instructions may not delay the interrupt. Thus, in the following instruction sequence:
   STI
   POP SS
   POP ESP
   interrupts may be recognized before the POP ESP executes, because STI also delays interrupts for one instruction.

```
            THEN #SS(selector);
    ELSE
        SS ← segment selector;
        SS ← segment descriptor;
    FI;
FI;
IF DS, ES, FS or GS is loaded with non-null selector;
THEN
    IF segment selector index is outside descriptor table limits
        OR segment is not a data or readable code segment
        OR ((segment is a data or nonconforming code segment)
            AND (both RPL and CPL > DPL))
                THEN #GP(selector);
        IF segment not marked present
            THEN #NP(selector);
    ELSE
        SegmentRegister ← segment selector;
        SegmentRegister ← segment descriptor;
    FI;
FI;
IF DS, ES, FS or GS is loaded with a null selector;
    THEN
        SegmentRegister ← segment selector;
        SegmentRegister ← segment descriptor;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If attempt is made to load SS register with null segment selector.
	If the destination operand is in a nonwritable segment.
	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.
#GP(selector)	If segment selector index is outside descriptor table limits.
	If the SS register is being loaded and the segment selector's RPL and the segment descriptor's DPL are not equal to the CPL.
	If the SS register is being loaded and the segment pointed to is a nonwritable data segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.
#SS(0)	If the current top of stack is not within the stack segment.
	If a memory operand effective address is outside the SS segment limit.
#SS(selector)	If the SS register is being loaded and the segment pointed to is marked not present.
#NP	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is marked not present.
#PF(fault-code)	If a page fault occurs.

#AC(0)	If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference is made while alignment checking is enabled.

53.14 POPA/POPAD—Pop All General-Purpose Registers

Opcode	Instruction	Description
61	POPA	Pop DI, SI, BP, BX, DX, CX, and AX
61	POPAD	Pop EDI, ESI, EBP, EBX, EDX, ECX, and EAX

Description

Pops doublewords (POPAD) or words (POPA) from the stack into the general-purpose registers. The registers are loaded in the following order: EDI, ESI, EBP, EBX, EDX, ECX, and EAX (if the operand-size attribute is 32) and DI, SI, BP, BX, DX, CX, and AX (if the operand-size attribute is 16). (These instructions reverse the operation of the PUSHA/PUSHAD instructions.) The value on the stack for the ESP or SP register is ignored. Instead, the ESP or SP register is incremented after each register is loaded.

The POPA (pop all) and POPAD (pop all double) mnemonics reference the same opcode. The POPA instruction is intended for use when the operand-size attribute is 16 and the POPAD instruction for when the operand-size attribute is 32. Some assemblers may force the operand size to 16 when POPA is used and to 32 when POPAD is used (using the operand-size override prefix [66H] if necessary). Others may treat these mnemonics as synonyms (POPA/POPAD) and use the current setting of the operand-size attribute to determine the size of values to be popped from the stack, regardless of the mnemonic used. (The D flag in the current code segment's segment descriptor determines the operand-size attribute.)

Operation

```
IF OperandSize = 32 (* instruction = POPAD *)
THEN
     EDI ← Pop();
     ESI ← Pop();
     EBP ← Pop();
     increment ESP by 4 (* skip next 4 bytes of stack *)
     EBX ← Pop();
     EDX ← Pop();
     ECX ← Pop();
     EAX ← Pop();
ELSE (* OperandSize = 16, instruction = POPA *)
     DI ← Pop();
     SI ← Pop();
     BP ← Pop();
```

```
    increment ESP by 2 (* skip next 2 bytes of stack *)
    BX ← Pop();
    DX ← Pop();
    CX ← Pop();
    AX ← Pop();
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#SS(0)	If the starting or ending stack address is not within the stack segment.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.

Real-Address Mode Exceptions

#SS	If the starting or ending stack address is not within the stack segment.

Virtual-8086 Mode Exceptions

#SS(0)	If the starting or ending stack address is not within the stack segment.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference is made while alignment checking is enabled.

53.15 POPF/POPFD—Pop Stack into EFLAGS Register

Opcode	Instruction	Description
9D	POPF	Pop top of stack into lower 16 bits of EFLAGS
9D	POPFD	Pop top of stack into EFLAGS

Description

Pops a doubleword (POPFD) from the top of the stack (if the current operand-size attribute is 32) and stores the value in the EFLAGS register or pops a word from the top of the stack (if the operand-size attribute is 16) and stores it in the lower 16 bits of the EFLAGS register (that is, the FLAGS register). (These instructions reverse the operation of the PUSHF/PUSHFD instructions.)

The POPF (pop flags) and POPFD (pop flags double) mnemonics reference the same opcode. The POPF instruction is intended for use when the operand-size attribute is 16 and the POPFD instruction for when the operand-size attribute is 32. Some assemblers may force the operand size to 16 when POPF is used and to 32 when POPFD is used. Others may treat these mnemonics as synonyms (POPF/POPFD) and use the current setting of the operand-size attribute to determine the size of values to be popped from the stack, regardless of the mnemonic used.

The effect of the POPF/POPFD instructions on the EFLAGS register changes slightly, depending on the mode of operation of the processor. When the processor is operating in protected mode at privilege level 0 (or in real-address mode, which is equivalent to privilege level 0), all the non-reserved flags in the EFLAGS register except the VIP, VIF, and VM flags can be modified. The VIP and VIF flags are cleared, and the VM flag is unaffected.

When operating in protected mode, with a privilege level greater than 0, but less than or equal to IOPL, all the flags can be modified except the IOPL field and the VIP, VIF, and VM flags. Here, the IOPL flags are unaffected, the VIP and VIF flags are cleared, and the VM flag is unaffected. The interrupt flag (IF) is altered only when executing at a level at least as privileged as the IOPL. If a POPF/POPFD instruction is executed with insufficient privilege, an exception does not occur, but the privileged bits do not change.

When operating in virtual-8086 mode, the I/O privilege level (IOPL) must be equal to 3 to use POPF/POPFD instructions and the VM, RF, IOPL, VIP, and VIF flags are unaffected. If the IOPL is less than 3, the POPF/POPFD instructions cause a general-protection exception (#GP).

See “EFLAGS Register”, for information about the EFLAGS registers.

Operation

```
IF VM=0 (* Not in Virtual-8086 Mode *)
    THEN IF CPL=0
        THEN
            IF OperandSize = 32;
                THEN
                    EFLAGS ← Pop();
                    (* All non-reserved flags except VIP, VIF, and VM can be modified; *)
                    (* VIP and VIF are cleared; VM is unaffected*)
                ELSE (* OperandSize = 16 *)
                    EFLAGS[15:0] ← Pop(); (* All non-reserved flags can be modified; *)
            FI;
        ELSE (* CPL > 0 *)
            IF OperandSize = 32;
                THEN
                    EFLAGS ← Pop()
                    (* All non-reserved bits except IOPL, VIP, and VIF can be modified; *)
                    (* IOPL is unaffected; VIP and VIF are cleared; VM is unaffected *)
                ELSE (* OperandSize = 16 *)
                    EFLAGS[15:0] ← Pop();
                    (* All non-reserved bits except IOPL can be modified *)
                    (* IOPL is unaffected *)
            FI;
    FI;
    ELSE  (* In Virtual-8086 Mode *)
        IF IOPL=3
            THEN IF OperandSize=32
                THEN
                    EFLAGS ← Pop()
                    (* All non-reserved bits except VM, RF, IOPL, VIP, and VIF *)
                    (* can be modified; VM, RF, IOPL, VIP, and VIF are unaffected *)
                ELSE
                    EFLAGS[15:0] ← Pop()
                    (* All non-reserved bits except IOPL can be modified *)
                    (* IOPL is unaffected *)
            FI;
            ELSE (* IOPL < 3 *)
                #GP(0);  (* trap to virtual-8086 monitor *)
        FI;
    FI;
FI;
```

Flags Affected

All flags except the reserved bits and the VM bit.

Protected Mode Exceptions

#SS(0)	If the top of stack is not within the stack segment.

#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.

Real-Address Mode Exceptions

#SS	If the top of stack is not within the stack segment.

Virtual-8086 Mode Exceptions

#GP(0)	If the I/O privilege level is less than 3. If an attempt is made to execute the POPF/POPFD instruction with an operand-size override prefix.
#SS(0)	If the top of stack is not within the stack segment.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference is made while alignment checking is enabled.

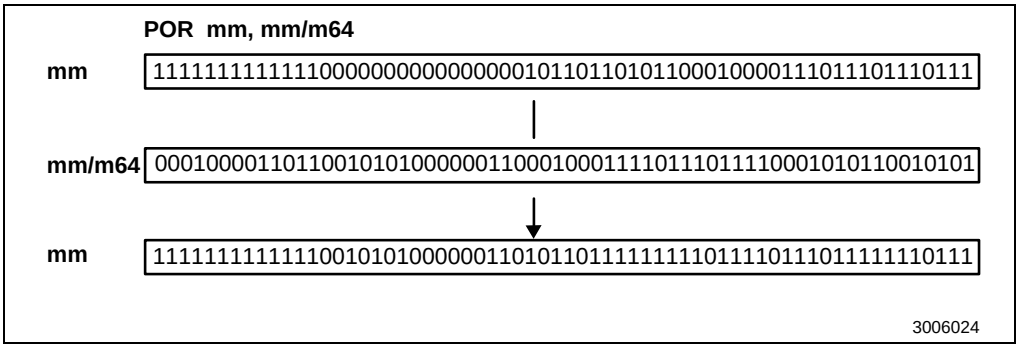
53.16 POR—Bitwise Logical OR

Opcode	Instruction	Description
0F EB /r	POR *mm, mm/m64*	OR quadword from mm/m64 to quadword in mm.

Description

Performs a bitwise logical OR operation on the quadword source (second) and destination (first) operands and stores the result in the destination operand location (see Figure 53-13). The source operand can be an MMX register or a quadword memory location; the destination operand must be an MMX register. Each bit of the result is made 0 if the corresponding bits of both operands are 0; otherwise the bit is set to 1.

Figure 53-13. Operation of the POR Instruction.



Operation

```
DEST ← DEST OR SRC;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.17 PSLLW/PSLLD/PSLLQ—Packed Shift Left Logical

Opcode	Instruction	Description
0F F1 /r	PSLLW *mm, mm/m64*	Shift words in *mm* left by amount specified in *mm/m64*, while shifting in zeros.
0F 71 /6, ib	PSLLW *mm, imm8*	Shift words in *mm* left by *imm8*, while shifting in zeros.
0F F2 /r	PSLLD *mm, mm/m64*	Shift doublewords in *mm* left by amount specified in *mm/m64*, while shifting in zeros.

0F 72 /6 ib	PSLLD *mm, imm8*	Shift doublewords in *mm* by *imm8*, while shifting in zeros.
0F F3 /r	PSLLQ *mm, mm/m64*	Shift *mm* left by amount specified in *mm/m64*, while shifting in zeros.
0F 73 /6 ib	PSLLQ *mm, imm8*	Shift *mm* left by Imm8, while shifting in zeros.

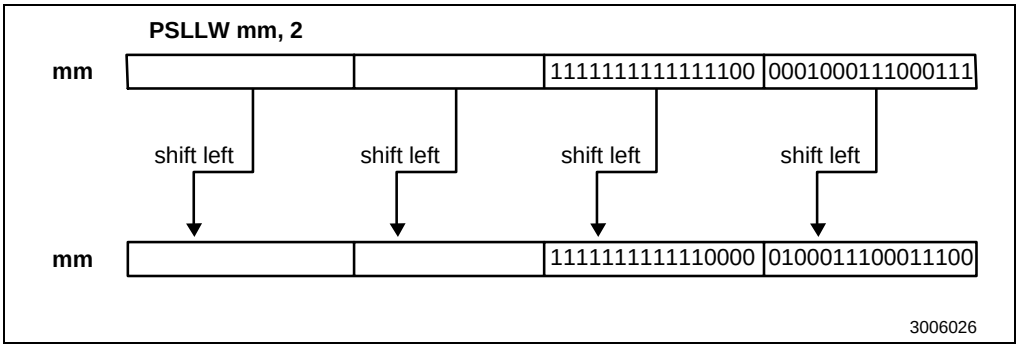
Description

Shifts the bits in the data elements (words, doublewords, or quadword) in the destination operand (first operand) to the left by the number of bits specified in the unsigned count operand (second operand). (See Figure 53-14.) The result of the shift operation is written to the destination operand. As the bits in the data elements are shifted left, the empty low-order bits are cleared (set to zero). If the value specified by the count operand is greater than 15 (for words), 31 (for doublewords), or 63 (for a quadword), then the destination operand is set to all zeros.

The destination operand must be an MMX register; the count operand can be either an MMX register, a 64-bit memory location, or an 8-bit immediate.

The PSLLW instruction shifts each of the four words of the destination operand to the left by the number of bits specified in the count operand; the PSLLD instruction shifts each of the two doublewords of the destination operand; and the PSLLQ instruction shifts the 64-bit quadword in the destination operand. As the individual data elements are shifted left, the empty low-order bit positions are filled with zeros.

Figure 53-14. Operation of the PSLLW Instruction



Operation

```
IF instruction is PSLLW
    THEN
        DEST(15..0) ← DEST(15..0) << COUNT;
        DEST(31..16) ← DEST(31..16) << COUNT;
        DEST(47..32) ← DEST(47..32) << COUNT;
        DEST(63..48) ← DEST(63..48) << COUNT;
    ELSE IF instruction is PSLLD
        THEN {
            DEST(31..0) ← DEST(31..0) << COUNT;
            DEST(63..32) ← DEST(63..32) << COUNT;
        ELSE  (* instruction is PSLLQ *)
            DEST ← DEST << COUNT;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.18 PSRAW/PSRAD—Packed Shift Right Arithmetic

Opcode	Instruction	Description
0F E1 /r	PSRAW *mm, mm/ m64*	Shift words in *mm* right by amount specified in *mm/m64* while shifting in sign bits.
0F 71 /4 ib	PSRAW *mm, imm8*	Shift words in *mm* right by *imm8* while shifting in sign bits
0F E2 /r	PSRAD *mm, mm/ m64*	Shift doublewords in *mm* right by amount specified in *mm/m64* while shifting in sign bits.
0F 72 /4 ib	PSRAD *mm, imm8*	Shift doublewords in *mm* right by *imm8* while shifting in sign bits.

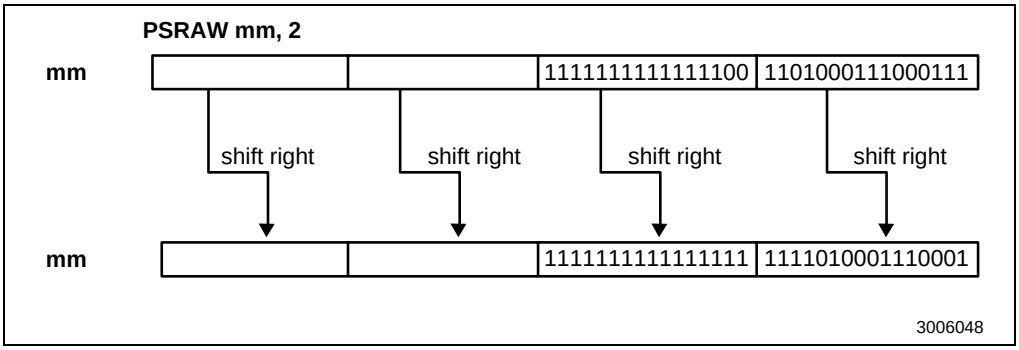
Description

Shifts the bits in the data elements (words or doublewords) in the destination operand (first operand) to the right by the amount of bits specified in the unsigned count operand (second operand). (See Figure 53-15.) The result of the shift operation is written to the destination operand. The empty high-order bits of each element are filled with the initial value of the sign bit of the data element. If the value specified by the count operand is greater than 15 (for words) or 31 (for doublewords), each destination data element is filled with the initial value of the sign bit of the element.

The destination operand must be an MMX register; the count operand (source operand) can be either an MMX register, a 64-bit memory location, or an 8-bit immediate.

The PSRAW instruction shifts each of the four words in the destination operand to the right by the number of bits specified in the count operand; the PSRAD instruction shifts each of the two doublewords in the destination operand. As the individual data elements are shifted right, the empty high-order bit positions are filled with the sign value.

Figure 53-15. Operation of the PSRAW Instruction



Operation

```
IF instruction is PSRAW
    THEN
        DEST(15..0) ← SignExtend (DEST(15..0) >> COUNT);
        DEST(31..16) ← SignExtend (DEST(31..16) >> COUNT);
        DEST(47..32) ← SignExtend (DEST(47..32) >> COUNT);
        DEST(63..48) ← SignExtend (DEST(63..48) >> COUNT);
    ELSE { (*instruction is PSRAD *)
        DEST(31..0) ← SignExtend (DEST(31..0) >> COUNT);
        DEST(63..32) ← SignExtend (DEST(63..32) >> COUNT);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.19 PSRLW/PSRLD/PSRLQ—Packed Shift Right Logical

Opcode	Instruction	Description
0F D1 /r	PSRLW *mm, mm/m64*	Shift words in *mm* right by amount specified in *mm/m64* while shifting in zeros.
0F 71 /2 ib	PSRLW *mm, imm8*	Shift words in *mm* right by *imm8*.
0F D2 /r	PSRLD *mm, mm/m64*	Shift doublewords in *mm* right by amount specified in *mm/m64* while shifting in zeros.
0F 72 /2 ib	PSRLD *mm, imm8*	Shift doublewords in *mm* right by *imm8*.
0F D3 /r	PSRLQ *mm, mm/m64*	Shift *mm* right by amount specified in *mm/m64* while shifting in zeros.
0F 73 /2 ib	PSRLQ *mm, imm8*	Shift *mm* right by *imm8* while shifting in zeros.

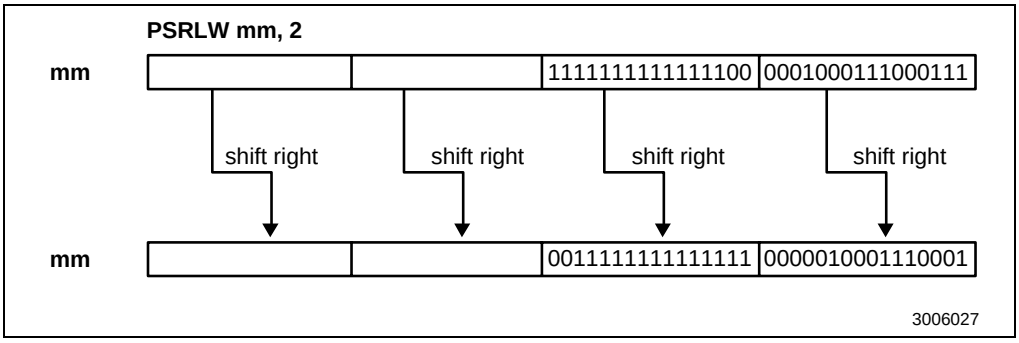
Description

Shifts the bits in the data elements (words, doublewords, or quadword) in the destination operand (first operand) to the right by the number of bits specified in the unsigned count operand (second operand). (See Figure 53-16.) The result of the shift operation is written to the destination operand. As the bits in the data elements are shifted right, the empty high-order bits are cleared (set to zero). If the value specified by the count operand is greater than 15 (for words), 31 (for doublewords), or 63 (for a quadword), then the destination operand is set to all zeros.

The destination operand must be an MMX register; the count operand can be either an MMX register, a 64-bit memory location, or an 8-bit immediate.

The PSRLW instruction shifts each of the four words of the destination operand to the right by the number of bits specified in the count operand; the PSRLD instruction shifts each of the two doublewords of the destination operand; and the PSRLQ instruction shifts the 64-bit quadword in the destination operand. As the individual data elements are shifted right, the empty high-order bit positions are filled with zeros.

Figure 53-16. Operation of the PSRLW Instruction



Operation

```
IF instruction is PSRLW
    THEN {
        DEST(15..0) ← DEST(15..0) >> COUNT;
        DEST(31..16) ← DEST(31..16) >> COUNT;
        DEST(47..32) ← DEST(47..32) >> COUNT;
        DEST(63..48) ← DEST(63..48) >> COUNT;
ELSE IF instruction is PSRLD
    THEN {
        DEST(31..0) ← DEST(31..0) >> COUNT;
        DEST(63..32) ← DEST(63..32) >> COUNT;
    ELSE  (* instruction is PSRLQ *)
        DEST ← DEST >> COUNT;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.

#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.20 PSUBB/PSUBW/PSUBD—Packed Subtract

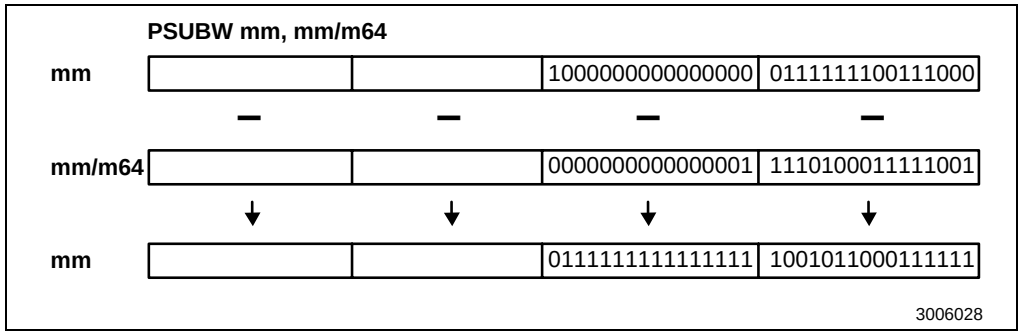
Opcode	Instruction	Description
0F F8 /r	PSUBB *mm, mm/m64*	Subtract packed bytes in *mm/m64* from packed bytes in *mm.*
0F F9 /r	PSUBW *mm, mm/m64*	Subtract packed words in*mm/m64* from packed words in *mm.*
0F FA /r	PSUBD *mm, mm/m64*	Subtract packed doublewords in *mm/m64* from packed doublewords in *mm.*

Description

Subtracts the individual data elements (bytes, words, or doublewords) of the source operand (second operand) from the individual data elements of the destination operand (first operand). (See Figure 53-17.) If the result of a subtraction exceeds the range for the specified data type (overflows), the result is wrapped around, meaning that the result is truncated so that only the lower (least significant) bits of the result are returned (that is, the carry is ignored).

The destination operand must be an MMX register; the source operand can be either an MMX register or a quadword memory location.

Figure 53-17. Operation of the PSUBW Instruction



The PSUBB instruction subtracts the bytes of the source operand from the bytes of the destination operand and stores the results to the destination operand. When an individual result is too large to be represented in 8 bits, the lower 8 bits of the result are written to the destination operand and therefore the result wraps around.

The PSUBW instruction subtracts the words of the source operand from the words of the destination operand and stores the results to the destination operand. When an individual result is too large to be represented in 16 bits, the lower 16 bits of the result are written to the destination operand and therefore the result wraps around.

The PSUBD instruction subtracts the doublewords of the source operand from the doublewords of the destination operand and stores the results to the destination operand. When an individual result is too large to be represented in 32 bits, the lower 32 bits of the result are written to the destination operand and therefore the result wraps around.

Note that like the integer SUB instruction, the PSUBB, PSUBW, and PSUBD instructions can operate on either unsigned or signed (two's complement notation) packed integers. Unlike the integer instructions, none of the MMX instructions affect the EFLAGS register. With MMX instructions, there are no carry or overflow flags to indicate when overflow has occurred, so the software must control the range of values or else use the "with saturation" MMX instructions.

Operation

```
IF instruction is PSUBB
    THEN
        DEST(7..0) ← DEST(7..0) – SRC(7..0);
        DEST(15..8) ← DEST(15..8) – SRC(15..8);
        DEST(23..16) ← DEST(23..16) – SRC(23..16);
        DEST(31..24) ← DEST(31..24) – SRC(31..24);
        DEST(39..32) ← DEST(39..32) – SRC(39..32);
        DEST(47..40) ← DEST(47..40) – SRC(47..40);
        DEST(55..48) ← DEST(55..48) – SRC(55..48);
        DEST(63..56) ← DEST(63..56) – SRC(63..56);
ELSEIF instruction is PSUBW
    THEN
        DEST(15..0) ← DEST(15..0) – SRC(15..0);
        DEST(31..16) ← DEST(31..16) – SRC(31..16);
        DEST(47..32) ← DEST(47..32) – SRC(47..32);
        DEST(63..48) ← DEST(63..48) – SRC(63..48);
    ELSE { (* instruction is PSUBD *)
        DEST(31..0) ← DEST(31..0) – SRC(31..0);
        DEST(63..32) ← DEST(63..32) – SRC(63..32);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

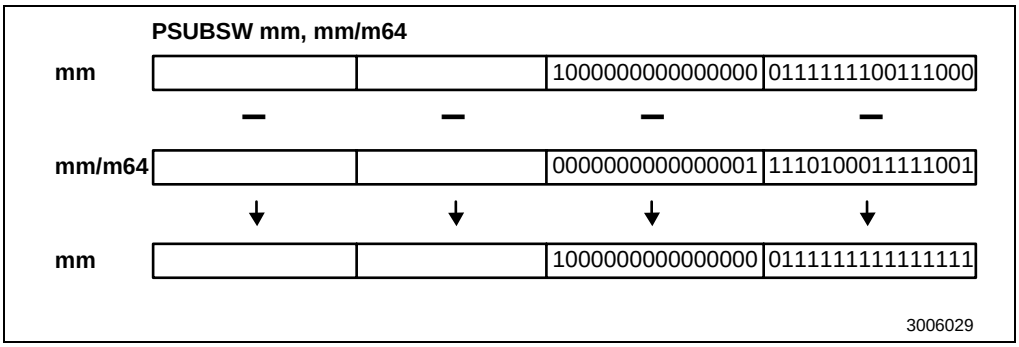
53.21 PSUBSB/PSUBSW—Packed Subtract with Saturation

Opcode	Instruction	Description
0F E8 /r	PSUBSB *mm, mm/m64*	Subtract signed packed bytes in *mm/m64* from signed packed bytes in *mm* and saturate.
0F E9 /r	PSUBSW *mm, mm/m64*	Subtract signed packed words in *mm/m64* from signed packed words in *mm* and saturate.

Description

Subtracts the individual signed data elements (bytes or words) of the source operand (second operand) from the individual signed data elements of the destination operand (first operand). (See Figure 53-18.) If the result of a subtraction exceeds the range for the specified data type, the result is saturated. The destination operand must be an MMX register; the source operand can be either an MMX register or a quadword memory location.

Figure 53-18. Operation of the PSUBSW Instruction



The PSUBSB instruction subtracts the signed bytes of the source operand from the signed bytes of the destination operand and stores the results to the destination operand. When an individual result is beyond the range of a signed byte (that is, greater than 7FH or less than 80H), the saturated byte value of 7FH or 80H, respectively, is written to the destination operand.

The PSUBSW instruction subtracts the signed words of the source operand from the signed words of the destination operand and stores the results to the destination operand. When an individual result is beyond the range of a signed word (that is, greater than 7FFFH or less than 8000H), the saturated word value of 7FFFH or 8000H, respectively, is written to the destination operand.

Operation

```
IF instruction is PSUBSB
    THEN
        DEST(7..0) ← SaturateToSignedByte(DEST(7..0) – SRC (7..0));
        DEST(15..8) ← SaturateToSignedByte(DEST(15..8) – SRC(15..8));
        DEST(23..16) ← SaturateToSignedByte(DEST(23..16) – SRC(23..16));
        DEST(31..24) ← SaturateToSignedByte(DEST(31..24) – SRC(31..24));
        DEST(39..32) ← SaturateToSignedByte(DEST(39..32) – SRC(39..32));
        DEST(47..40) ← SaturateToSignedByte(DEST(47..40) – SRC(47..40));
        DEST(55..48) ← SaturateToSignedByte(DEST(55..48) – SRC(55..48));
        DEST(63..56) ← SaturateToSignedByte(DEST(63..56) – SRC(63..56))
    ELSE (* instruction is PSUBSW *)
        DEST(15..0) ← SaturateToSignedWord(DEST(15..0) – SRC(15..0));
        DEST(31..16) ← SaturateToSignedWord(DEST(31..16) – SRC(31..16));
        DEST(47..32) ← SaturateToSignedWord(DEST(47..32) – SRC(47..32));
        DEST(63..48) ← SaturateToSignedWord(DEST(63..48) – SRC(63..48));
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.

#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

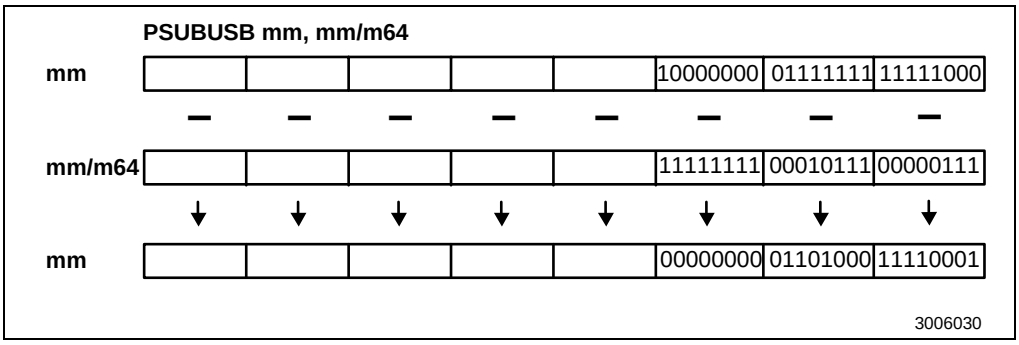
53.22 PSUBUSB/PSUBUSW—Packed Subtract Unsigned with Saturation

Opcode	Instruction	Description
0F D8 /r	PSUBUSB *mm, mm/m64*	Subtract unsigned packed bytes in *mm/m64* from unsigned packed bytes in *mm* and saturate.
0F D9 /r	PSUBUSW *mm, mm/m64*	Subtract unsigned packed words in *mm/m64* from unsigned packed words in *mm* and saturate.

Description

Subtracts the individual unsigned data elements (bytes or words) of the source operand (second operand) from the individual unsigned data elements of the destination operand (first operand). (See Figure 53-19.) If the result of an individual subtraction exceeds the range for the specified unsigned data type, the result is saturated. The destination operand musts be an MMX register; the source operand can be either an MMX register or a quadword memory location.

Figure 53-19. Operation of the PSUBUSB Instruction



The PSUBUSB instruction subtracts the unsigned bytes of the source operand from the unsigned bytes of the destination operand and stores the results to the destination operand. When an individual result is less than zero (a negative value), the saturated unsigned byte value of 00H is written to the destination operand.

The PSUBUSW instruction subtracts the unsigned words of the source operand from the unsigned words of the destination operand and stores the results to the destination operand. When an individual result is less than zero (a negative value), the saturated unsigned word value of 0000H is written to the destination operand.

Operation

```
IF instruction is PSUBUSB
    THEN
        DEST(7..0) ← SaturateToUnsignedByte (DEST(7..0 – SRC (7..0) );
        DEST(15..8) ← SaturateToUnsignedByte ( DEST(15..8) – SRC(15..8) );
        DEST(23..16) ← SaturateToUnsignedByte (DEST(23..16) – SRC(23..16) );
        DEST(31..24) ← SaturateToUnsignedByte (DEST(31..24) – SRC(31..24) );
        DEST(39..32) ← SaturateToUnsignedByte (DEST(39..32) – SRC(39..32) );
        DEST(47..40) ← SaturateToUnsignedByte (DEST(47..40) – SRC(47..40) );
        DEST(55..48) ← SaturateToUnsignedByte (DEST(55..48) – SRC(55..48) );
        DEST(63..56) ← SaturateToUnsignedByte (DEST(63..56) – SRC(63..56) );
    ELSE { (* instruction is PSUBUSW *)
        DEST(15..0) ← SaturateToUnsignedWord (DEST(15..0) – SRC(15..0) );
        DEST(31..16) ← SaturateToUnsignedWord (DEST(31..16) – SRC(31..16) );
        DEST(47..32) ← SaturateToUnsignedWord (DEST(47..32) – SRC(47..32) );
        DEST(63..48) ← SaturateToUnsignedWord (DEST(63..48) – SRC(63..48) );
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.

#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

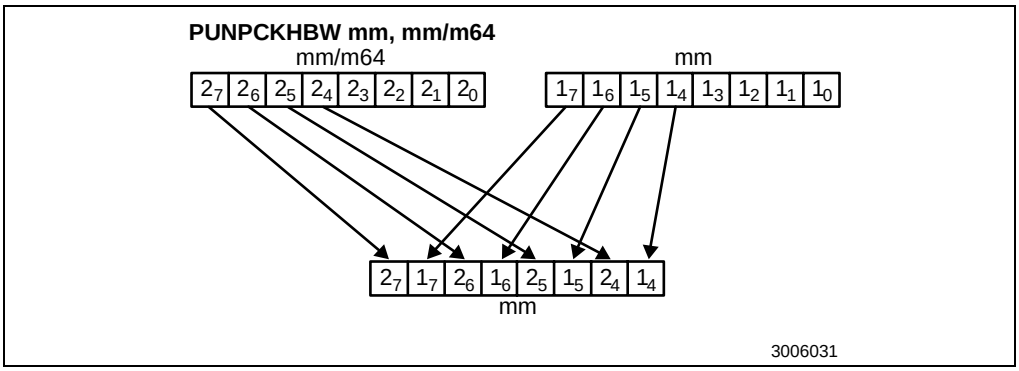
53.23 PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ—Unpack High Packed Data

Opcode	Instruction	Description
0F 68 /r	PUNPCKHBW *mm, mm/m64*	Interleave high-order bytes from *mm* and *mm/m64* into *mm.*
0F 69 /r	PUNPCKHWD *mm, mm/m64*	Interleave high-order words from *mm* and *mm/m64* into *mm.*
0F 6A /r	PUNPCKHDQ *mm, mm/m64*	Interleave high-order doublewords from *mm* and *mm/m64* into *mm.*

Description

Unpacks and interleaves the high-order data elements (bytes, words, or doublewords) of the destination operand (first operand) and source operand (second operand) into the destination operand (see Figure 53-20). The low-order data elements are ignored. The destination operand must be an MMX register; the source operand may be either an MMX register or a 64-bit memory location. When the source data comes from a memory operand, the full 64-bit operand is accessed from memory, but the instruction uses only the high-order 32 bits.

Figure 53-20. High-Order Unpacking and Interleaving of Bytes With the PUNPCKHBW Instruction



The PUNPCKHBW instruction interleaves the four high-order bytes of the source operand and the four high-order bytes of the destination operand and writes them to the destination operand.

The PUNPCKHWD instruction interleaves the two high-order words of the source operand and the two high-order words of the destination operand and writes them to the destination operand.

The PUNPCKHDQ instruction interleaves the high-order doubleword of the source operand and the high-order doubleword of the destination operand and writes them to the destination operand.

If the source operand is all zeros, the result (stored in the destination operand) contains zero extensions of the high-order data elements from the original value in the destination operand. With the PUNPCKHBW instruction the high-order bytes are zero extended (that is, unpacked into unsigned words), and with the PUNPCKHWD instruction, the high-order words are zero extended (unpacked into unsigned doublewords).

Operation

```
IF instruction is PUNPCKHBW
    THEN
        DEST(7..0) ← DEST(39..32);
        DEST(15..8) ← SRC(39..32);
        DEST(23..16) ← DEST(47..40);
        DEST(31..24) ← SRC(47..40);
        DEST(39..32) ← DEST(55..48);
        DEST(47..40) ← SRC(55..48);
        DEST(55..48) ← DEST(63..56);
        DEST(63..56) ← SRC(63..56);
ELSE IF instruction is PUNPCKHW
    THEN
        DEST(15..0) ← DEST(47..32);
        DEST(31..16) ← SRC(47..32);
        DEST(47..32) ← DEST(63..48);
        DEST(63..48) ← SRC(63..48);
    ELSE (* instruction is PUNPCKHDQ *)
        DEST(31..0) ← DEST(63..32)
        DEST(63..32) ← SRC(63..32);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.

#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

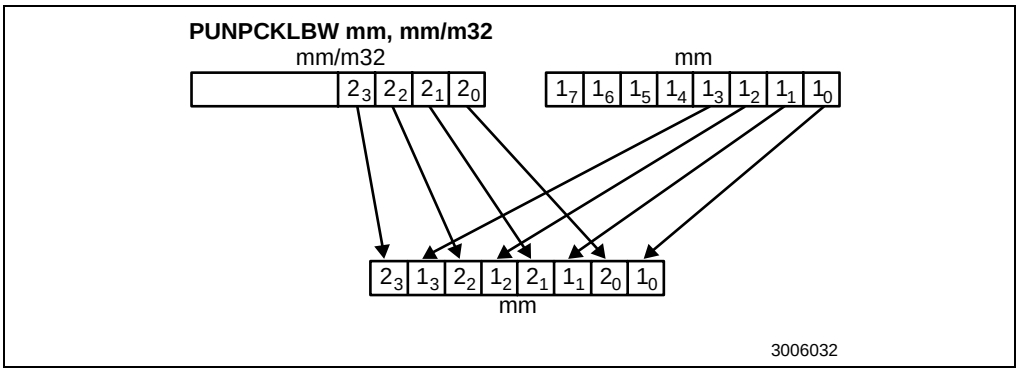
53.24 PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ—Unpack Low Packed Data

Opcode	Instruction	Description
0F 60 /r	PUNPCKLBW *mm, mm/m32*	Interleave low-order bytes from *mm* and *mm/m64* into *mm.*
0F 61 /r	PUNPCKLWD *mm, mm/m32*	Interleave low-order words from *mm* and *mm/m64* into *mm.*
0F 62 /r	PUNPCKLDQ *mm, mm/m32*	Interleave low-order doublewords from *mm* and *mm/m64* into *mm.*

Description

Unpacks and interleaves the low-order data elements (bytes, words, or doublewords) of the destination and source operands into the destination operand (see Figure 53-21). The destination operand must be an MMX register; the source operand may be either an MMX register or a memory location. When source data comes from an MMX register, the upper 32 bits of the register are ignored. When the source data comes from a memory, only 32-bits are accessed from memory.

Figure 53-21. Low-Order Unpacking and Interleaving of Bytes With the PUNPCKLBW Instruction



The PUNPCKLBW instruction interleaves the four low-order bytes of the source operand and the four low-order bytes of the destination operand and writes them to the destination operand.

The PUNPCKLWD instruction interleaves the two low-order words of the source operand and the two low-order words of the destination operand and writes them to the destination operand.

The PUNPCKLDQ instruction interleaves the low-order doubleword of the source operand and the low-order doubleword of the destination operand and writes them to the destination operand.

If the source operand is all zeros, the result (stored in the destination operand) contains zero extensions of the high-order data elements from the original value in the destination operand. With the PUNPCKLBW instruction the low-order bytes are zero extended (that is, unpacked into unsigned words), and with the PUNPCKLWD instruction, the low-order words are zero extended (unpacked into unsigned doublewords).

Operation

```
IF instruction is PUNPCKLBW
    THEN
        DEST(63..56) ← SRC(31..24);
        DEST(55..48) ← DEST(31..24);
        DEST(47..40) ← SRC(23..16);
        DEST(39..32) ← DEST(23..16);
        DEST(31..24) ← SRC(15..8);
        DEST(23..16) ← DEST(15..8);
        DEST(15..8) ← SRC(7..0);
        DEST(7..0) ← DEST(7..0);
ELSE IF instruction is PUNPCKLWD
    THEN
        DEST(63..48) ← SRC(31..16);
        DEST(47..32) ← DEST(31..16);
        DEST(31..16) ← SRC(15..0);
        DEST(15..0) ← DEST(15..0);
    ELSE (* instruction is PUNPCKLDQ *)
        DEST(63..32) ← SRC(31..0);
        DEST(31..0) ← DEST(31..0);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.25 PUSH—Push Word or Doubleword Onto the Stack

Opcode	Instruction	Description
FF /6	PUSH *r/m16*	Push *r/m16*
FF /6	PUSH *r/m32*	Push *r/m32*
50+*rw*	PUSH *r16*	Push *r16*
50+*rd*	PUSH *r32*	Push *r32*
6A	PUSH *imm8*	Push *imm8*
68	PUSH *imm16*	Push *imm16*

68	PUSH *imm32*	Push *imm32*
0E	PUSH CS	Push CS
16	PUSH SS	Push SS
1E	PUSH DS	Push DS
06	PUSH ES	Push ES
0F A0	PUSH FS	Push FS
0F A8	PUSH GS	Push GS

Description

Decrements the stack pointer and then stores the source operand on the top of the stack. The address-size attribute of the stack segment determines the stack pointer size (16 bits or 32 bits), and the operand-size attribute of the current code segment determines the amount the stack pointer is decremented (2 bytes or 4 bytes). For example, if these address- and operand-size attributes are 32, the 32-bit ESP register (stack pointer) is decremented by 4 and, if they are 16, the 16-bit SP register is decremented by 2.(The B flag in the stack segment's segment descriptor determines the stack's address-size attribute, and the D flag in the current code segment's segment descriptor, along with prefixes, determines the operand-size attribute and also the address-size attribute of the source operand.) Pushing a 16-bit operand when the stack address-size attribute is 32 can result in a misaligned the stack pointer (that is, the stack pointer is not aligned on a doubleword boundary).

The PUSH ESP instruction pushes the value of the ESP register as it existed before the instruction was executed. Thus, if a PUSH instruction uses a memory operand in which the ESP register is used as a base register for computing the operand address, the effective address of the operand is computed before the ESP register is decremented.

In the real-address mode, if the ESP or SP register is 1 when the PUSH instruction is executed, the processor shuts down due to a lack of stack space. No exception is generated to indicate this condition.

Intel Architecture Compatibility

For Intel Architecture processors from the Intel 286 on, the PUSH ESP instruction pushes the value of the ESP register as it existed before the instruction was executed. (This is also true in the real-address and virtual-8086 modes.) For the Intel 8086 processor, the PUSH SP instruction pushes the new value of the SP register (that is the value after it has been decremented by 2).

Operation

```
IF StackAddrSize = 32
THEN
    IF OperandSize = 32
        THEN
            ESP ← ESP – 4;
            SS:ESP ← SRC; (* push doubleword *)
        ELSE (* OperandSize = 16*)
            ESP ← ESP – 2;
            SS:ESP ← SRC; (* push word *)
    FI;
ELSE (* StackAddrSize = 16*)
    IF OperandSize = 16
        THEN
            SP ← SP – 2;
             SS:SP ← SRC; (* push word *)
        ELSE (* OperandSize = 32*)
            SP ← SP – 4;
            SS:SP ← SRC; (* push doubleword *)
    FI;
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register is used to access memory and it contains a null segment selector.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
	If the new value of the SP or ESP register is outside the stack segment limit.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.

53.26 PUSHA/PUSHAD—Push All General-Purpose Registers

Opcode	Instruction	Description
60	PUSHA	Push AX, CX, DX, BX, original SP, BP, SI, and DI
60	PUSHAD	Push EAX, ECX, EDX, EBX, original ESP, EBP, ESI, and EDI

Description

Pushes the contents of the general-purpose registers onto the stack. The registers are stored on the stack in the following order: EAX, ECX, EDX, EBX, ESP (original value), EBP, ESI, and EDI (if the current operand-size attribute is 32) and AX, CX, DX, BX, SP (original value), BP, SI, and DI (if the operand-size attribute is 16). (These instructions perform the reverse operation of the POPA/POPAD instructions.) The value pushed for the ESP or SP register is its value before prior to pushing the first register (see the “Operation” section below).

The PUSHA (push all) and PUSHAD (push all double) mnemonics reference the same opcode. The PUSHA instruction is intended for use when the operand-size attribute is 16 and the PUSHAD instruction for when the operand-size attribute is 32. Some assemblers may force the operand size to 16 when PUSHA is used and to 32 when PUSHAD is used. Others may treat these mnemonics as synonyms (PUSHA/PUSHAD) and use the current setting of the operand-size attribute to determine the size of values to be pushed from the stack, regardless of the mnemonic used.

In the real-address mode, if the ESP or SP register is 1, 3, or 5 when the PUSHA/PUSHAD instruction is executed, the processor shuts down due to a lack of stack space. No exception is generated to indicate this condition.

Operation

```
IF OperandSize = 32 (* PUSHAD instruction *)
    THEN
        Temp ← (ESP);
        Push(EAX);
        Push(ECX);
        Push(EDX);
        Push(EBX);
        Push(Temp);
        Push(EBP);
        Push(ESI);
        Push(EDI);
    ELSE (* OperandSize = 16, PUSHA instruction *)
        Temp ← (SP);
        Push(AX);
        Push(CX);
        Push(DX);
        Push(BX);
        Push(Temp);
        Push(BP);
        Push(SI);
        Push(DI);
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#SS(0)	If the starting or ending stack address is outside the stack segment limit.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.

Real-Address Mode Exceptions

#GP	If the ESP or SP register contains 7, 9, 11, 13, or 15.

Virtual-8086 Mode Exceptions

#GP(0)	If the ESP or SP register contains 7, 9, 11, 13, or 15.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference is made while alignment checking is enabled.

53.27 PUSHF/PUSHFD—Push EFLAGS Register onto the Stack

Opcode	Instruction	Description
9C	PUSHF	Push lower 16 bits of EFLAGS
9C	PUSHFD	Push EFLAGS

Description

Decrements the stack pointer by 4 (if the current operand-size attribute is 32) and pushes the entire contents of the EFLAGS register onto the stack, or decrements the stack pointer by 2 (if the operand-size attribute is 16) and pushes the lower 16 bits of the EFLAGS register (that is, the FLAGS register) onto the stack. (These instructions reverse the operation of the POPF/POPFD instructions.) When copying the entire EFLAGS register to the stack, the VM and RF flags (bits 16 and 17) are not copied; instead, the values for these flags are cleared in the EFLAGS image stored on the stack. See “EFLAGS Register”, for information about the EFLAGS registers.

The PUSHF (push flags) and PUSHFD (push flags double) mnemonics reference the same opcode. The PUSHF instruction is intended for use when the operand-size attribute is 16 and the PUSHFD instruction for when the operand-size attribute is 32. Some assemblers may force the operand size to 16 when PUSHF is used and to 32 when PUSHFD is used. Others may treat these mnemonics as synonyms (PUSHF/PUSHFD) and use the current setting of the operand-size attribute to determine the size of values to be pushed from the stack, regardless of the mnemonic used.

When in virtual-8086 mode and the I/O privilege level (IOPL) is less than 3, the PUSHF/PUSHFD instruction causes a general protection exception (#GP).

In the real-address mode, if the ESP or SP register is 1, 3, or 5 when the PUSHA/PUSHAD instruction is executed, the processor shuts down due to a lack of stack space. No exception is generated to indicate this condition.

Operation

```
IF (PE=0) OR (PE=1 AND ((VM=0) OR (VM=1 AND IOPL=3)))
(* Real-Address Mode, Protected mode, or Virtual-8086 mode with IOPL equal to 3 *)
     THEN
          IF OperandSize = 32
              THEN
                   push(EFLAGS AND 00FCFFFFH);
                   (* VM and RF EFLAG bits are cleared in image stored on the stack*)
              ELSE
                   push(EFLAGS); (* Lower 16 bits only *)
          FI;
     ELSE (* In Virtual-8086 Mode with IOPL less than 0 *)
          #GP(0); (* Trap to virtual-8086 monitor *)
FI;
```

Flags Affected

None.

Protected Mode Exceptions

#SS(0) If the new value of the ESP register is outside the stack segment boundary.

#PF(fault-code) If a page fault occurs.

#AC(0)	If an unaligned memory reference is made while the current privilege level is 3 and alignment checking is enabled.

Real-Address Mode Exceptions

None.

Virtual-8086 Mode Exceptions

#GP(0)	If the I/O privilege level is less than 3.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If an unaligned memory reference is made while alignment checking is enabled.

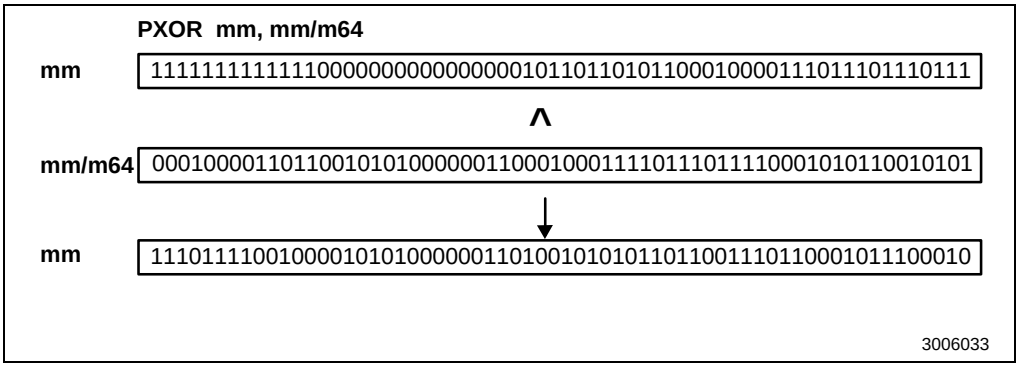
53.28 PXOR—Logical Exclusive OR

Opcode	Instruction	Description
0F EF /r	PXOR *mm, mm/m64*	XOR quadword from *mm/m64* to quadword in *mm.*

Description

Performs a bitwise logical exclusive-OR (XOR) operation on the quadword source (second) and destination (first) operands and stores the result in the destination operand location (see Figure 53-22). The source operand can be an MMX register or a quadword memory location; the destination operand must be an MMX register. Each bit of the result is 1 if the corresponding bits of the two operands are different; each bit is 0 if the corresponding bits of the operands are the same.

Figure 53-22. Operation of the PXOR Instruction



Operation

```
DEST ← DEST XOR SRC;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.

Virtual-8086 Mode Exceptions

#GP	If any part of the operand lies outside of the effective address space from 0 to FFFFH.
#UD	If EM in CR0 is set.
#NM	If TS in CR0 is set.
#MF	If there is a pending FPU exception.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made.